Créer un dépôt Git

Dans cette nouvelle leçon, nous allons utiliser nos premières commandes Git afin de créer un dépôt Git de manière pratique.

Pour rappel, il existe deux façons de créer un dépôt Git : on peut soit initialiser un dépôt Git à partir d'un répertoire déjà existant, soit cloner un dépôt Git déjà existant.

Créer un dépôt Git à partir d'un répertoire existant

Lorsqu'on démarre avec Git, on a souvent déjà des projets en cours stockés localement sur notre machine ou sur serveur distant et pour lesquels on aimerait implémenter un système de gestion de version.

Dans ce cas là, nous allons pouvoir importer l'ensemble des ressources d'un projet dans Git. Pour la suite de cette leçon, je vais créer un répertoire "projet-git" qui se trouve sur mon bureau et qui contient deux fichiers texte vides "fichier1.txt" et "README.txt". Ce répertoire va me servir de base pour les exemples qui vont suivre (ce sera le répertoire importé).

Je vous invite à créer le même répertoire sur votre machine. Vous pouvez le faire soit à la main, soit en utilisant la ligne de commandes comme ci-dessous (attention, toute mon installation et mon système sont en anglais, il est possible que vous ayez à remplacer "desktop" par "bureau" entre autres) :

```
pierres-macbook-pro:~ pierre$
pierres-macbook-pro:~ pierre$ cd desktop #On se place sur le bureau
pierres-macbook-pro:desktop pierre$ mkdir projet-git #Crée un dossier "projet-git" sur le bureau
pierres-macbook-pro:desktop pierre$ cd projet-git #Se place dans le dossier
pierres-macbook-pro:projet-git pierre$ touch fichier1.txt #Crée un fichier texte dans le dossier
pierres-macbook-pro:projet-git pierre$ touch README.txt #Crée un autre fichier texte
pierres-macbook-pro:projet-git pierre$ ls #Affiche le contenu du dossier
README.txt      fichier1.txt
```

La commande cd sert à de placer dans un répertoire. Dès qu'on est sur le bureau, on utilise mkdir pour créer un répertoire vide qu'on appelle "projet-git". On se place dans ce répertoire et on crée deux fichiers texte grâce à la commande Bash touch. On utilise enfin ls pour afficher le contenu du répertoire et s'assurer que tout a bien fonctionné.

Pour initialiser un dépôt Git, on utilise ensuite la commande git init comme ci-dessous. Cela crée un sous répertoire .git qui contient un ensemble de fichiers qui vont permettre à un dépôt Git de fonctionner.

```
pierres-macbook-pro:projet-git pierre$ git init #Initialise un dépôt Git
Initialized empty Git repository in /Users/pierre/Desktop/projet-git/.git/
```

versionné aucun fichier (nous n'avons ajouté aucun fichier du répertoire en base).

On peut utiliser ici la commande git status pour déterminer l'état des fichiers de notre répertoire. Cette commande est extrêmement utile et c'est une de celles que j'utilise le plus personnellement :

```
pierres-macbook-pro:projet-git pierre$ git init #Initialise un dépôt Git
Initialized empty Git repository in /Users/pierre/Desktop/projet-git/.git/
pierres-macbook-pro:projet-git pierre$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.txt
        fichier1.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Ici, git status nous informe que notre projet possède deux fichiers qui ne sont pas sous suivi de version ("untracked") et qui sont les fichiers "README.txt" et "ficiher1.txt". Il nous dit aussi qu'aucun fichier n'a été validé ("commit") en base pour le moment ni ajouté pour validation. La commande git statuts nous informe également sur la branche sur laquelle on se trouve ("master" ici). Nous reparlerons des branches plus tard.

L'étape suivante va donc ici être d'indexer nos fichiers afin qu'ils puissent ensuite être validés, c'est-à-dire ajoutés en base et qu'on puisse ainsi avoir un premier historique de version.

Pour indexer des fichiers, on utilise la commande git add. On peut lui passer un nom de fichier pour indexer le fichier en question, le nom d'un répertoire pour indexer tous les fichiers du répertoire d'un coup ou encore un "fileglob" pour ajouter tous les fichiers correspondant au schéma fourni.

Les fileglobs utilisent les extension de chemin de fichier. Grosso-modo, cela signifie que certains caractères comme * et ? vont posséder une signification spéciale et nous permettre de créer des schémas de correspondances. Le caractère * par exemple correspond à n'importe quel caractère. Lorsque j'écris git add *.txt, je demande finalement à Git d'ajouter à l'index tous les fichiers du projet qui possèdent une extension .txt, quelque soit leur nom.

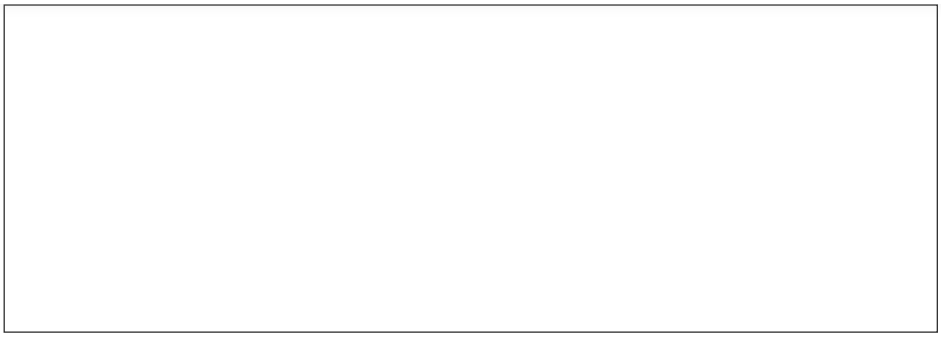
Si on relance une commande git status, on obtient les informations suivantes :

git status nous dit qu'on a maintenant deux nouveaux fichiers ajoutés à l'index. La commande git add permet en fait de faire plusieurs choses : elle permet d'indexer des fichiers déjà sous suivi de version et de placer sous suivi des fichiers non suivi (en plus de les indexer).

Ici, on est certains que nos deux nouveaux fichiers ont bien été ajoutés à l'index puisqu'ils apparaissent dans la section "changes to be committed" ("modifications à valider").

Pour valider ces fichiers et les ajouter en base, on va maintenant utiliser la commande git commit comme cela :

Lorsqu'on utilise git commit sans argument, une nouvelle fenêtre s'ouvre en utilisant l'éditeur



Ici, on nous demande d'ajouter un message avec notre commit. Bien documenter chaque commit permet aux auteurs et aux différents contributeurs à un projet de rapidement comprendre les modifications et de pouvoir les valider. C'est une part essentielle de Git. Ici, j'ajoute simplement le message "Version initiale du projet".

Une fois le message entré, si votre éditeur est bien VIM, il faut appuyer sur la touche esc" pour sortir du mode insertion puis taper :wq et entree pour valider et quitter ou :x et entree ou tout simplement ZZ.

Une fois sorti de VIM, un message s'affiche avec des informations sur le commit effectué.

On nous informe ici qu'on se situe sur la branche master, qu'il s'agit du premier commit (root-commit) et on nous donne sa somme de contrôle (4ed866e) qui permet de l'identifier de manière unique. On nous dit également que deux fichiers ont été modifiés et que 0 lignes ont été ajoutées ou supprimées dans ces fichiers.

Si on effectue à nouveau un git status, voici le message renvoyé :

Git nous informe désormais qu'il n'y a plus aucun fichier à vider, ce qui signifie que tous les fichiers sont sous suivi de version et sont enregistrés en base et qu'aucune modification n'a été apportée à ces fichiers depuis le dernier commit.

Cloner un dépôt Git

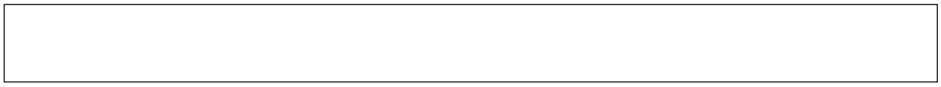
La deuxième façon de démarrer un dépôt Git est de cloner localement un dépôt Git déjà existant. Pour cela, on va utiliser la commande Git clone.

En pratique, dans la grande majorité des cas, nous clonerons des dépôts Git distants, c'est-à-dire des dépôts hébergés sur serveur distants pour pouvoir contribuer à ces projets.

Cependant, nous pouvons également cloner des dépôts locaux. Nous parlerons des dépôts distants et apprendrons à les cloner lorsqu'on abordera GitHub. Pour le moment, contentons nous d'essayer de cloner notre dépôt "projet-git" tout juste créé.

Pour cela, on va se placer sur le bureau. Comme je suis pour le moment situé dans mon répertoire "projet-git", j'utilise la commande Bash cd .. pour atteindre le répertoire parent (c'est-à-dire mon bureau).

créer le clone sur le bureau par simplicité et on va donc simplement passer un nom à nouveau. Appelons le clone “projet-git-2” par exemple comme ceci :



On peut cd dans le répertoire du projet et effectuer un ultime git status pour s’assurer de l’état des fichiers du répertoire :

Comme vous pouvez le voir, le dépôt a bien été cloné puisque les fichiers du répertoire projet-git-2 sont déjà bien tous sous suivi de version et sont stockés en base.

Précédent Suivant

Laisser un commentaire

Vous devez vous connecter pour publier un commentaire.